Sequence-Structured Decision Learning for Branch-and-Bound in Mixed-Integer Linear Programming

Anonymous authors
Paper under double-blind review

Abstract

Branch-and-bound (BnB) solves mixed-integer linear programs by constructing and exploring a search tree. Node selection and branching are two recurrent decisions that directly determine the tree. Learning-based approaches typically optimize either decision in isolation and condition on a mostly local view, leaving the coupled evolution of the search tree insufficiently modeled. This paper proposes Seq-BnB, a sequence-structured framework that learns branching and node selection from a shared global representation of the evolving BnB process. Seq-BnB serializes node creations, node updates, node-selection events, and branching events into typed tokens, producing a causal event sequence of the search trajectory. A self-attention encoder summarizes this global search history, a bipartite graph neural network encodes the current LP relaxation, and context-conditioned decision heads score branching variables and open nodes. To reduce distribution shift between expert demonstrations and learned policies, Seq-BnB is trained with DAgger-style data aggregation. On four MILP benchmark families, Seq-BnB consistently processes fewer BnB nodes than heuristic and learning-based baselines, with strong solution times on both standard and larger transfer instances. While this paper evaluates Seq-BnB instantiation for branching and node selection, it accommodates additional solver decisions, such as cut selection, by introducing new action tokens and decision heads.

Keywords: mixed-integer linear programming; branch-and-bound; learn to optimize; imitation learning; graph neural networks

1 Introduction

Mixed-integer linear programming (MILP) is a central modeling tool in operations research and decision analytics, with applications ranging from network design and scheduling to allocation and logistics (Nemhauser & Wolsey, 1988; Wolsey, 1998). The classical Branch-and-Bound (BnB) algorithm solves MILPs by recursively decomposing the feasible region into a dynamically growing search tree (Land & Doig, 1960; Dakin, 1965). Modern branch-and-cut solvers enrich this framework with presolving, cutting planes, primal heuristics, and sophisticated tree management (Achterberg, 2009; Gamrath et al., 2020). Within the process, two recurrent decisions directly govern the tree growth: node selection chooses the open subproblem to process next, and branching partitions that subproblem into new descendants (Linderoth & Savelsbergh, 1999; Achterberg et al., 2005). Their interaction influences when incumbents are discovered, how bounds improve, which subtrees are pruned, and ultimately how much search is required.

A large body of work has studied these decision rules. Classical branching rules include strong branching, pseudocost branching, and reliability branching; search rules include best-estimate search and related node-selection strategies (Achterberg et al., 2005; Achterberg, 2009; Lodi & Zarpellon, 2017). Learning-based methods replace these hand-designed scores with policies learned from solver data. For branching, they imitate strong branching or incorporate search-tree information to rank candidate variables at the current node (Khalil

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2016; Alvarez et al., 2017; Gasse et al., 2019; Zarpellon et al., 2021; Gupta et al., 2022; Bengio et al., 2021). For node selection, they rank or compare open nodes in the active set (He et al., 2014; Labassi et al., 2022). Reinforcement learning and retrospective trajectory construction were also explored for branching and BnB search control (Etheve et al., 2020; Parsonson et al., 2023). Most methods learn one decision type at a time: branching models choose variables, whereas node-selection models choose open nodes.

Recent work moves beyond isolated solver decisions. Multi-task and foundation-model approaches learn shared MILP representations for different prediction or control tasks, while unified BnB policies jointly consider node selection and variable selection (Cai et al., 2025; Li et al., 2025; Du et al., 2025). Other recent methods incorporate branching history or sequential structure, or explicitly represent the BnB tree for node selection (Lin et al., 2022; Zhang et al., 2026; 2025). These studies demonstrate the value of shared representations and coupled solver decisions. They also leave open how to represent the chronological history of node-selection and branching events as a common context for both decisions.

The remaining gap is therefore not only whether multiple solver decisions are learned together, but how the evolving BnB process is represented. Branching and node selection act on different candidate sets but modify the same search tree. A branching policy observes fractional variables at the current LP relaxation, while a node-selection policy observes open subproblems in the active tree. Even when the two decisions are jointly modeled, an effective policy must account for the trajectory that has produced the current tree, including previous branching choices, search-order decisions, incumbent updates, bound changes, and the evolving distribution of open nodes. Without such a unified historical representation, a locally favorable decision for one component may lead to an unfavorable downstream tree when evaluated together with the other component.

The sequential nature of BnB further strengthens this issue. BnB is a policy-induced process: deployed decisions change the nodes, candidate variables, and search histories encountered later. This distributional issue is well known in imitation learning and sequential prediction (Ross et al., 2011), and it is particularly important inside BnB because early actions reshape the subsequent search states. Joint decision learning therefore requires two ingredients: a shared representation of the evolving search process and training data collected from states induced by the learned policy.

This motivates us to propose Seq-BnB in this paper. Seq-BnB provides a shared, causally available search-state representation for coupled BnB decisions. Rather than treating the BnB tree only as a collection of locally scored candidates, Seq-BnB incrementally serializes the evolving tree and its decision trajectory into typed tokens. Node tokens describe created or updated subproblems, while action tokens record node-selection and branching events together with global search statistics. A self-attention encoder (Vaswani et al., 2017) summarizes the resulting global context. For the current LP relaxation, a bipartite graph neural network encodes local variable–constraint structure, following the representation used in learning-based branching (Gasse et al., 2019). Cross-attention combines global tree context with local candidate features and produces both branching and node-selection scores. To mitigate distribution shift caused by deploying learned decisions, Seq-BnB is trained with a DAgger-style data aggregation procedure that progressively incorporates states visited by the learned policy.

The sequential formulation is not tied to only branching and node selection decisions. These two form the minimal coupled decision set studied in this paper because they determine where and how the BnB tree grows. More broadly, learning can also be applied to other solver components, including primal solution prediction, cut selection, primal heuristics, heuristic scheduling, and solver configuration (Nair et al., 2020; Sonnerat et al., 2021; Paulus et al., 2022; Deza & Khalil, 2023; Chmiela et al., 2021). The typed-token representation offers a common interface for such actions: additional solver events can be recorded as new token types and equipped with corresponding prediction heads. This perspective is also complementary to emerging generalist and foundation-model approaches for MILP (Li et al., 2025). We regard this extensibility as a design implication of the representation, while empirically validating the branching-and-node-selection instantiation in this paper.

To summarize, our contributions are as follows:

- We formulate branching and node selection as coupled tree-construction decisions in BnB. Instead of treating branching as a local variable-ranking problem and node selection as a separate open-node-ranking problem, we model them as two recurrent actions that jointly induce the search tree.
- We introduce a causal event-sequence representation for BnB. Seq-BnB serializes node creations, node-state updates, node-selection actions, and branching actions into typed tokens. The resulting sequence provides a common global state for heterogeneous solver decisions while preserving the online execution order of BnB. This representation differs from local LP-graph encodings by explicitly modeling search history, tree evolution, and previous solver actions.
- We develop a joint learning architecture for branching and node selection. A shared self-attention encoder summarizes the global event sequence, a bipartite graph neural network represents the current LP relaxation, and context-conditioned decision heads score both branching variables and open nodes. The two heads are trained with a joint objective and DAgger-style data aggregation, allowing the policy to learn from states induced by its own deployment inside the solver.

We provide computational studies on four MILP benchmark families. Seq-BnB consistently processes fewer BnB nodes than heuristic and learning-based baselines on both standard and larger transfer instances. Ablation studies show that removing sequence context from the branching head, learning only one of the two decisions, or training without DAgger substantially degrades performance. These results support the central claim that shared sequential context improves coupled BnB decision making.

2 Background and Related Work

A MILP has continuous and integer variables coupled by linear constraints. BnB solves LP relaxations over subproblems and prunes nodes that are infeasible, integer feasible, or unable to improve the incumbent. At step t, the solver selects an open node $n_t \in \mathcal{O}_t$; if branching is needed, it selects a fractional integer variable x_{j_t} and creates child subproblems. Because branching changes future open nodes and node selection determines where branching is applied, the two decisions jointly induce the evolving tree. Classical policies include strong branching, pseudocost branching, and best-estimate search (Achterberg et al., 2005; Linderoth & Savelsbergh, 1999).

Learning-based MILP research has mainly targeted individual BnB decisions (Lodi & Zarpellon, 2017; Bengio et al., 2021). Branching policies progressed from hand-crafted features and strong-branching imitation to bipartite graph representations and reinforcement learning (Khalil et al., 2016; Alvarez et al., 2017; Gasse et al., 2019; Parsonson et al., 2023). Recent work also models branching history: T-BranT uses a tree-aware transformer (Lin et al., 2022), while Mamba-Branching models sequences of branching states and decisions (Zhang et al., 2026). Node-selection policies learn search orders or compare open nodes (He et al., 2014; Labassi et al., 2022); TRGNN represents the evolving tree as a tripartite graph for node selection (Zhang et al., 2025). These history- and tree-aware methods learn one decision type at a time.

Shared MILP representations have also been investigated across solver tasks and problem classes (Cai et al., 2025; Li et al., 2025). More closely related, Du et al. (Du et al., 2025) jointly integrate node and variable selection, recognizing their interdependence. Seq-BnB focuses on a shared chronological sequence of node creations, updates, node-selection actions, and branching actions as context for both decisions. This representation combines the evolving search history with the current LP graph and supports on-policy data aggregation. Further background and comparisons appear in Appendix A.

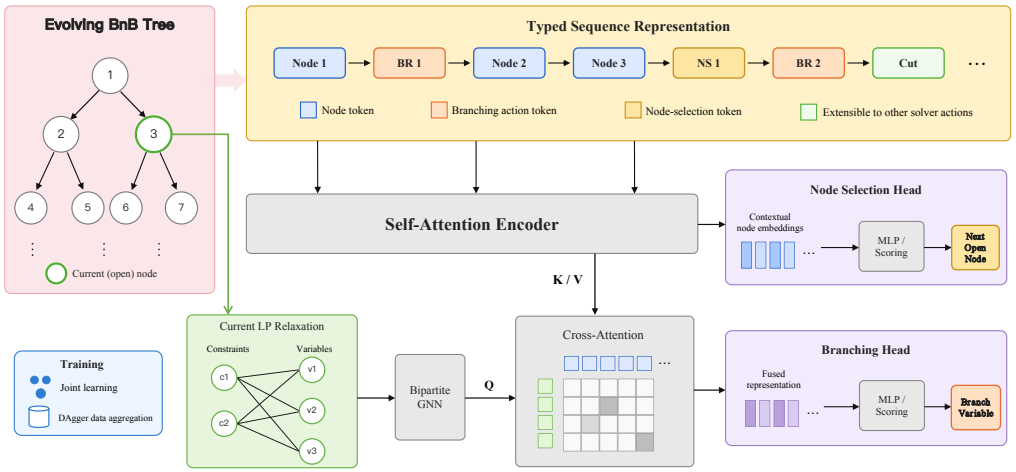


Figure 1: Architecture of Seq-BnB. The evolving BnB tree and solver actions are serialized into typed tokens and encoded by a self-attention module. Local LP relaxations are encoded by a bipartite GNN, and cross-attention conditions branching and node-selection decisions on the global search context.

3 Seq-BnB Framework

Seq-BnB has three components: a sequence-structured global representation of the evolving BnB tree, a local graph encoder for the current LP relaxation, and decision heads that query the shared encoded context. In the present instantiation, the heads correspond to branching and node selection, the two coupled decisions that generate the search tree. Figure 1 summarizes the architecture.

3.1 Sequential Representation of BnB

Let $\mathcal{Z}_t = (z_1, z_2, \ldots, z_{L_t})$ denote the sequence available immediately before decision step t. Each token belongs to one of three types:

$$z_i \in \{\text{Node}, \text{NS}, \text{BR}\}, \tag{1}$$

where Node records the creation or state update of a BnB node, NS records a node-selection decision, and BR records a branching decision. This event-based representation is aligned with the solver's natural execution order and can be updated online as the search evolves. Conceptually, the sequence is an incremental serialization of the generated search tree together with the actions that produced its growth, providing a global carrier of information shared across solver decisions. In implementation, the bounded window described below retains recent history and all currently open nodes, avoiding the claim that every historical token must be retained indefinitely.

The token vocabulary is modular. For example, a future extension that learns cut selection could introduce a Cut action token recording the cut-selection state and selected cut, and attach a cut-scoring head to the same encoded context. Thus, the representation is designed as an interface for decision learning over the BnB process; the current paper focuses on the NS and BR tokens and heads.

For a BnB node u, the node-token feature vector is

$$\mathbf{x}^{\text{node}}(u) = [\ell(u), c(u), d(u), \mathbf{1}_{\text{visited}}(u), \mathbf{1}_{\text{fathomed}}(u), \mathbf{1}_{\text{incumbent}}(u), s(u), g_t], \tag{2}$$

where $\ell(u)$ is the LP lower bound, $c(u)$ is the estimate value, $d(u)$ is depth, $s(u)$ is the creation or visit order, and g_t is a global gap indicator. The binary indicators record whether the node has been processed, fathomed, or associated with an incumbent update.

After the node-selection decision has selected n_t from $\mathcal{O}_t$ and the LP relaxation at n_t has been solved, we record the node-selection event by constructing

$$\mathbf{x}_t^{\mathrm{ns}} = [g_t, |\mathcal{O}_t|, t, d(n_t), \ell(n_t), c(n_t), \mathbf{s}^{\mathrm{node}}(n_t)], \tag{3}$$

where $\mathbf{s}^{\mathrm{node}}(n_t)$ is a pooled summary of the local graph representation at n_t.

For branching, if variable x_{j_t} is selected at n_t, we construct

$$\mathbf{x}_t^{\mathrm{br}} = [g_t, |\mathcal{O}_t|, t, d(n_t), \ell(n_t), c(n_t), \mathbf{h}_{j_t}^{\mathrm{var}}(n_t)], \tag{4}$$

where $\mathbf{h}_{j_t}^{\mathrm{var}}(n_t)$ is the local embedding of the selected variable.

Online availability and causal construction. Seq-BnB is constructed online and uses only information that is available when a solver decision is scored. At a node-selection decision, the solver has an open-node set $\mathcal{O}_t$, the incumbent information available at that time, global bounds, and the stored attributes of open nodes. Seq-BnB scores each open node using its node token in the current sequence. It does not solve LP relaxations for unprocessed open nodes only for the purpose of node selection. The local LP graph representation is computed only after the selected node n_t is processed and its LP relaxation is solved. Therefore, the summary $\mathbf{s}^{\mathrm{node}}(n_t)$ in the node-selection action token in Eq. equation 3 is a record of a completed selection event and is not used to score n_t itself. Similarly, the variable embedding $\mathbf{h}_{j_t}^{\mathrm{var}}(n_t)$ in the branching action token in Eq. equation 4 is available only after the LP relaxation at n_t has been solved and the branching candidate set has been formed.

Equivalently, the sequence is causal, i.e., before a decision is scored, it contains only tokens whose corresponding solver events have already occurred. The token recording the current node-selection decision is appended only after that node has been selected, and the token recording the current branching decision is appended only after the branching variable has been selected. These action tokens may condition future decisions, but they cannot affect the scores that produced the actions themselves. The global gap feature g_t is also computed from the incumbent and global bound available at the current decision time; before the first incumbent is found, a fixed missing-value encoding is used. Table 4 in Appendix B summarizes the information used by each token type.

Each token is embedded by combining a type embedding and a type-specific feature projection:

$$\mathbf{e}_i = \mathbf{e}_{\mathrm{type}}(\kappa(z_i)) + \mathrm{MLP}_{\kappa(z_i)}(\mathbf{x}(z_i)) \in \mathbb{R}^d, \tag{5}$$

where $\kappa(z_i)$ is the token type. To control memory and attention cost, we keep a recent history window of at most $L_{\max}$ tokens and force the tokens associated with currently open nodes to remain available. This choice keeps recent search history while preserving the candidates needed for node selection.

3.2 Global Sequence Encoding and Local Graph Encoding

The embedded token matrix is

$$\mathbf{E}_t = [\mathbf{e}_1, \ldots, \mathbf{e}_{L_t}]^\top \in \mathbb{R}^{L_t \times d}. \tag{6}$$

We pass $\mathbf{E}_t$ through a self-attention encoder to obtain contextual search-state representations:

$$\mathbf{H}_t = \mathrm{SelfAttention}(\mathbf{E}_t, M_{\mathrm{pad}}) \in \mathbb{R}^{L_t \times d}, \tag{7}$$

where M_{pad} masks invalid positions. The node-order feature $s(u)$ and action-step feature t provide explicit temporal information to the encoder, allowing it to relate current decisions to earlier tree events.

For the current decision node n_t, we encode the LP relaxation as a bipartite graph

$$\mathcal{G}(n_t) = (\mathcal{C}(n_t), \mathcal{V}(n_t), \mathcal{E}(n_t)), \tag{8}$$

where constraint nodes and variable nodes are connected by nonzero constraint coefficients. A graph neural network produces variable embeddings

$$\mathbf{H}^{\mathrm{var}}(n_t) = \mathrm{GNN}(\mathcal{G}(n_t)) \in \mathbb{R}^{|\mathcal{V}(n_t)| \times d}. \tag{9}$$

The embedding of candidate variable x_j is denoted by $\mathbf{h}_j^{\mathrm{var}}(n_t)$. We also define a node summary

$$\mathbf{s}^{\mathrm{node}}(n_t) = \mathrm{Pool}(\mathbf{H}^{\mathrm{var}}(n_t)), \tag{10}$$

which is used in node-selection tokens.

3.3 Context-Conditioned Decision Heads

For branching, Seq-BnB uses each candidate variable embedding as a query and the sequence representation as keys and values:

$$\widetilde{\mathbf{h}}_j^{\mathrm{var}} = \mathrm{CrossAttention}(\mathbf{h}_j^{\mathrm{var}}(n_t), \mathbf{H}_t). \tag{11}$$

The branching score is

$$\hat{y}_j^{\mathrm{branch}} = \mathrm{MLP}_{\mathrm{branch}}([\mathbf{h}_j^{\mathrm{var}}(n_t); \widetilde{\mathbf{h}}_j^{\mathrm{var}}]). \tag{12}$$

The variable with the largest score is selected for branching.

For node selection, each open node $u \in \mathcal{O}_t$ has a corresponding node token in the sequence. Let $\mathbf{h}_u^{\mathrm{node}}$ be its contextual representation in $\mathbf{H}_t$. Seq-BnB scores open nodes by

$$\hat{y}_u^{\mathrm{ns}} = \mathrm{MLP}_{\mathrm{ns}}(\mathbf{h}_u^{\mathrm{node}}), \qquad u \in \mathcal{O}_t, \tag{13}$$

then chooses

$$n_t^* = \arg\max_{u \in \mathcal{O}_t} \hat{y}_u^{\mathrm{ns}}. \tag{14}$$

The two heads share the same sequence encoder. This sharing is central to the formulation: node selection and branching are evaluated against a common representation of the tree generated so far, rather than against two disconnected local states. The representation can capture the current gap, the distribution of open nodes, recent tree-growth patterns, and the timing of incumbent updates. Additional action heads could query the same sequence context when extending Seq-BnB to other solver decisions.

4 Training

4.1 DAgger Data Aggregation

Imitating expert trajectories alone (Pomerleau, 1991) can lead to distribution shift because learned decisions change later search states. Seq-BnB therefore uses DAgger (Ross et al., 2011). Branching labels come from strong branching. For node selection, each training MILP is solved offline to obtain a reference optimum $x^\star$. An open node is compatible if its branching bounds still contain $x^\star$; among compatible nodes, the oracle prefers the deepest node, with deterministic bound-based tie-breaking. If no compatible open node remains, it selects by a deterministic lower-bound rule. The reference solution is used for training labels only, never as an inference feature. Appendix C gives the exact compatibility and tie-breaking formulas.

Starting from expert trajectories, each DAgger round deploys the current policy in the solver, labels the resulting states with the two experts, aggregates them, and retrains the joint policy. This exposes training to the distribution of states created by learned decisions. Algorithm 1 summarizes the procedure.

4.2 Joint Loss

The two decision heads share the sequence encoder and minimize

$$\mathcal{L} = \mathcal{L}_{\mathrm{branch}} + \lambda \mathcal{L}_{\mathrm{ns}}, \tag{15}$$

where each term is a cross-entropy loss over the corresponding candidate set and λ controls the relative node-selection weight. The complete losses are in Appendix C.

Algorithm 1 DAgger training for Seq-BnB

Collect expert trajectories and initialize $\widetilde{\mathcal{D}}_0 \leftarrow \mathcal{D}_0$.
Train initial policy $\hat{\pi}_0$ on $\widetilde{\mathcal{D}}_0$.
for $k = 1, \ldots, K$ do
    Run the solver with policy $\hat{\pi}_{k-1}$ and collect unlabeled decision states $\mathcal{D}_k^{\text{raw}}$.
    Label branching states with strong branching and node-selection states with the oracle.
    Add the labeled states to the aggregate data set: $\widetilde{\mathcal{D}}_k \leftarrow \widetilde{\mathcal{D}}_{k-1} \cup \mathcal{D}_k$.
    Train $\hat{\pi}_k$ on $\widetilde{\mathcal{D}}_k$.
end for
return $\hat{\pi}_K$.

5 Computational Experiments

5.1 Experimental Setup

We evaluate four standard MILP families: Set Covering (Balas & Ho, 2009), Combinatorial Auction (Leyton-Brown et al., 2000), Capacitated Facility Location (Cornuéjols et al., 1991), and Max Independent Set (Bergman et al., 2016; Alvarez et al., 2017). Each has 100 standard test instances and 100 larger transfer instances; sizes are reported in Appendix D.

We compare against heuristic and learning-based combinations. We use L2B (Gasse et al., 2019) and L2C (Labassi et al., 2022) as graph-based, imitation-learning single-decision baselines: their local representations and learning settings provide closer reference points for assessing the contribution of Seq-BnB's shared search-history representation. T-BranT (Lin et al., 2022) and Mamba-Branching (Zhang et al., 2026) introduce different history-aware branching architectures, while TRGNN (Zhang et al., 2025) uses a tripartite tree representation and reinforcement learning for node selection; we discuss these methods in Section 2 rather than treating them as controlled baselines of our joint architecture. Du et al.'s joint node-and-variable selection method (Du et al., 2025) is closely related, but we could not locate publicly available code or complete a reliable reimplementation, so it is not included in the experimental comparison. Branching baselines include strong branching (SB), pseudocost branching (PSB), the GNN branching model L2B (Gasse et al., 2019), and Retro (Parsonson et al., 2023). Node-selection baselines include Oracle, Best Estimate Search, RankNet (Song et al., 2018), and L2C (Labassi et al., 2022). The expert SB+Oracle is included as a reference and is not counted when selecting the best nonexpert result.

All methods use SCIP 7.0.3, with cuts only at the root and no restarts as in prior branching studies (Karzan et al., 2009; Fischetti & Monaci, 2012; Khalil et al., 2016), and a 3,600-second limit. Time and processed-node counts are shifted geometric means (shift 1); unsolved instances contribute their final time and processed-node count. Each learning-based method uses 120,000 training samples. Seq-BnB starts with 10,000 expert samples and collects about 10,000 per round over 11 DAgger rounds. We use Adam, batch size 64, learning rate 10^{-5}, $L_{\max} = 500$, and $\lambda = 0.1$. Training uses one NVIDIA A100 GPU; inference uses one thread and fixed seeds. The code used for training and evaluation will be made publicly available upon publication.

5.2 Main Results

Table 1 reports the standard test sets. Seq-BnB processes the fewest nodes on all four families. It is fastest on Set Covering and Max Independent Set; on Combinatorial Auction and Facility Location its time remains close to the fastest method. Relative to L2B+Estimate, the strongest learning-based branching combination in these experiments, Seq-BnB uses fewer nodes in every standard family.

Table 2 evaluates larger instances from the same families. Seq-BnB again processes the fewest nodes in all four and achieves the fastest time on Facility Location and Max Independent Set. On Set Covering and Combinatorial Auction, PSB+Estimate and L2B+Estimate are faster, respectively. This distinction matters: processed nodes reflect search effort within the time limit, whereas wall-clock time also includes sequence construction, model inference, and varying LP costs. The transfer results show that the search benefit persists as instance

Table 1: Standard test results (100 instances per family).

Method	Set Covering		Comb. Auction		Facility Location		Max Ind. Set	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
SB+Oracle	9.01	20.89	59.34	183.28	82.99	131.58	101.37	30.44
PSB+Estimate	3.73	37.25	10.98	886.54	41.33	226.81	9.88	101.93
RankNet+PSB	3.87	37.49	16.52	1247.04	28.70	256.24	11.42	86.91
L2C+PSB	3.60	38.86	29.75	1806.76	30.80	265.38	7.78	156.95
L2B+Estimate	2.99	32.06	10.13	292.23	28.60	180.19	6.45	38.10
Retro+Estimate	6.36	136.16	18.44	1124.61	35.36	230.98	12.27	144.39
Seq-BnB	2.81	28.62	10.61	252.34	30.22	164.16	6.35	36.54

Time is in seconds. Time and Nodes are shifted geometric means; bold marks the best nonexpert method. Full results including wins are in Appendix E.

Table 2: Results on larger transfer instances (100 instances per family).

Method	Set Covering		Comb. Auction		Facility Location		Max Ind. Set	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
SB+Oracle	121.58	234.79	3363.43	2074.32	2304.62	207.62	2795.54	183.65
PSB+Estimate	25.66	667.39	2174.09	21105.59	748.19	1390.19	1217.84	13517.28
RankNet+PSB	40.38	802.18	1228.58	37844.01	2113.60	4398.48	644.66	11997.45
L2C+PSB	45.72	696.74	1526.97	66897.30	1407.08	3611.90	601.47	16559.32
L2B+Estimate	28.97	462.74	618.79	11771.63	747.34	1475.62	459.58	6085.88
Retro+Estimate	514.17	7766.10	3591.17	27588.27	1162.38	1740.44	2171.13	18567.67
Seq-BnB	37.33	402.63	790.99	5134.41	621.68	1249.23	412.42	1179.97

Time is in seconds. Time and Nodes are shifted geometric means; bold marks the best nonexpert method. Full results including wins are in Appendix E.

size grows, while inference overhead remains a practical target for improvement. Complete results including wins appear in Appendix E.

5.3 Ablations

Table 3 tests the central design choices on standard instances. Removing sequence context from the branching head increases both time and processed nodes in every family, showing the value of history-conditioned branching. The BR-only variant pairs learned branching with Estimate, and NS-only pairs learned node selection with PSB. The integrated policy outperforms both, supporting the value of learning the two coupled decisions together. Training without DAgger uses static expert data of the same total sample size and performs worse throughout, consistent with the policy-induced state-distribution motivation.

Appendix F gives the complete tables and branching imitation accuracy. Appendix G analyzes $L_{\max}$ and λ: the default settings attain the best time and processed-node count among the tested values on all four families.

6 Conclusion

This paper presents Seq-BnB, a sequence-structured framework for jointly learning branching and node selection in MILP branch-and-bound. These decisions are coupled because node selection determines where search continues and branching determines how that region expands. Seq-BnB represents their common consequence, the evolving search tree and its decision trajectory, through typed tokens processed by a shared self-attention encoder. A bipartite GNN supplies local LP structure, cross-attention connects global context to local candidates, and DAgger-style data aggregation reduces the distribution shift induced by deploying learned policies inside the solver.

Across four MILP benchmark families, Seq-BnB consistently reduces the number of processed BnB nodes and often improves solution time. The joint-learning ablation supports the value of modeling branching and node selection together, while transfer experiments show that the sequence context remains useful on larger instances. Beyond the two actions evaluated here, the sequence representation provides a natural interface for integrating ad-

Table 3: Core ablations on standard test instances (Time/Nodes).

Method	Set Covering		Comb. Auction		Facility Location		Max Ind. Set	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
w/o Seq (branch head)	3.51	31.94	11.83	281.63	41.56	238.42	7.48	42.73
BR-only	3.04	34.24	12.18	306.54	39.56	225.63	6.92	39.84
NS-only	3.80	32.25	27.47	684.73	40.88	236.94	7.89	68.74
w/o DAgger	5.36	43.21	18.21	397.63	49.37	374.35	8.64	49.51
Seq-BnB	2.81	28.62	10.61	252.34	30.22	164.16	6.35	36.54

Time is in seconds; Time and Nodes are shifted geometric means. Detailed tables and branching accuracy are in Appendix F.

ditional solver decisions, such as cut selection or primal-heuristic scheduling, through new token types and decision heads. Evaluating such extensions and reducing runtime overhead through cached token embeddings, batched candidate scores, or lighter sequence encoders are directions for future work.

AI use statement

During the preparation of this work, the authors used ChatGPT (OpenAI) to assist with language refinement, readability improvement, and the organization of manuscript content. After using this tool, the authors carefully reviewed and edited the manuscript as needed and take full responsibility for the content of the published article.

References

Tobias Achterberg. Scip: Solving constraint integer programs. Mathematical Programming Computation, 1:1–41, 2009.

Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. Operations Research Letters, 33(1):42–54, 2005.

Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. INFORMS Journal on Computing, 29(1):185–195, 2017.

Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. In Combinatorial Optimization, pp. 37–60. Springer, 2009.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research, 290(2):405–421, 2021.

David Bergman, Andre A. Cire, Willem-Jan Van Hoeve, and John Hooker. Decision Diagrams for Optimization. Springer, 2016.

Junyang Cai, Taoan Huang, and Bistra Dilkina. Multi-task representation learning for mixed integer linear programming. In International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 134–151. Springer, 2025.

Antonia Chmiela, Elias B. Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. In Advances in Neural Information Processing Systems, volume 34, pp. 24235–24246, 2021.

Gérard Cornuéjols, Ranjani Sridharan, and Jean-Michel Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. European Journal of Operational Research, 50(3):280–297, 1991.

Robert J. Dakin. A tree-search algorithm for mixed integer programming problems. The Computer Journal, 8(3):250–255, 1965.

Arnaud Deza and Elias B. Khalil. Machine learning for cutting planes in integer programming: A survey. In Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, pp. 6592–6600, 2023.

Shuhan Du, Junbo Tong, and Wenhui Fan. Learning efficient branch-and-bound for solving mixed integer linear programs. Applied Soft Computing, 172:112863, 2025.

Marc Etheve, Zacharie Alès, Côme Bissuel, Olivier Juan, and Safia Kedad-Sidhoum. Reinforcement learning for variable selection in a branch and bound algorithm. In Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 176–185. Springer, 2020.

Matteo Fischetti and Michele Monaci. Branching on nonchimerical fractionalities. Operations Research Letters, 40(3):159–164, 2012.

Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Florian Matter, Matthias Miltenberger, Erik M"uhmer, Benjamin M"uller, Marc E. Pfetsch, Franziska Schl"osser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP optimization suite 7.0. ZIB Report 20-10, Zuse Institute Berlin, 2020.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 15580–15592, 2019.

Prateek Gupta, Elias B. Khalil, Didier Chételat, Maxime Gasse, Yoshua Bengio, Andrea Lodi, and M. Pawan Kumar. Lookback for learning to branch. Transactions on Machine Learning Research, 2022.

He He, Hal Daumé III, and Jason Eisner. Learning to search in branch and bound algorithms. In Advances in Neural Information Processing Systems, volume 27, 2014.

Fatma Kılınç Karzan, George L. Nemhauser, and Martin W. P. Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. Mathematical Programming Computation, 1(4):249–293, 2009.

Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016.

Abdel Ghani Labassi, Didier Chételat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. In Proceedings of the 36th International Conference on Neural Information Processing Systems, pp. 32000–32010, 2022.

Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. Econometrica, 28(3):497–520, 1960.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In Proceedings of the 2nd ACM Conference on Electronic Commerce, pp. 66–76, 2000.

Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In The Thirteenth International Conference on Learning Representations, pp. 88590–88638, 2025.

Jiacheng Lin, Jialin Zhu, Huangang Wang, and Tao Zhang. Learning to branch with tree-aware branching transformers. Knowledge-Based Systems, 252:109455, 2022. doi: 10.1016/j.knosys.2022.109455.

Jeff T. Linderoth and Martin W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing, 11(2):173–187, 1999.

Andrea Lodi and Giulia Zarpellon. On learning and branching: A survey. TOP, 25(2): 207–236, 2017.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. arXiv preprint arXiv:2012.13349, 2020.

George L. Nemhauser and Laurence A. Wolsey. Integer and Combinatorial Optimization. John Wiley & Sons, New York, NY, USA, 1988.

Christopher W. F. Parsonson, Alexandre Laterre, and Thomas D. Barrett. Reinforcement learning for branch-and-bound optimisation using retrospective trajectories. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pp. 4061–4069, 2023.

Max B. Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris J. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In Proceedings of the 39th International Conference on Machine Learning, volume 162, pp. 17584–17600. PMLR, 2022.

Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. Neural Computation, 3(1):88–97, 1991.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Jialin Song, Ravi Lanka, Albert Zhao, Akshay Bhatnagar, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. arXiv preprint arXiv:1804.00846, 2018.

Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. arXiv preprint arXiv:2107.10201, 2021.

Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive cut selection in mixed-integer linear programming. Open Journal of Mathematical Optimization, 4: 1–31, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 6000–6010, 2017.

Laurence A. Wolsey. Integer Programming. John Wiley & Sons, New York, NY, USA, 1998.

Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. Parameterizing branch-and-bound search trees to learn branching policies. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pp. 3931–3939, 2021.

Ce Zhang, Bin Zhang, and Guoliang Fan. Towards better branching policies: Leveraging the sequential nature of branch-and-bound tree. In The Fourteenth International Conference on Learning Representations, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b2f56a345fe0ce337df3bdfb31e2b350-Abstract-Conference.html.

Sijia Zhang, Shuli Zeng, Shaoang Li, Feng Wu, and Xiangyang Li. Learning to select nodes in branch and bound with sufficient tree representation. In The Thirteenth International Conference on Learning Representations, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/82f625a28d822d2748b9f5c4f9a89bb9-Abstract-Conference.html.

A Additional Background and Related Work

A.1 MILP

This paper considers a MILP of the form

$$\min\{c^\top x : Ax \le b,\ x_j \in \mathbb{Z} \text{ for } j \in \mathcal{I}\}, \tag{16}$$

where x contains continuous and integer variables. BnB solves equation 16 by repeatedly solving LP relaxations over subproblems. If the LP solution at a node is infeasible, integer feasible, or cannot improve the incumbent, the node is fathomed. Otherwise, the solver branches on a fractional integer variable and creates child nodes with additional bounds or disjunctive constraints.

At each search step t, the solver maintains an open-node set $\mathcal{O}_t$, an incumbent solution if one has been found, and global bounds. Two tree-construction decisions are made repeatedly. The node-selection policy chooses the next node $n_t \in \mathcal{O}_t$ to process, thereby choosing the current region of exploration. The branching policy then chooses a candidate fractional variable x_{j_t} at n_t and creates child subproblems, thereby determining the local expansion of that region. Because each branching action changes the future open-node set and each node-selection action determines where subsequent branching is applied, the two policies jointly induce the evolving search tree.

A.2 BnB Search and Solver Decisions

BnB is the backbone of exact MILP solving. Its classical form recursively partitions the feasible region and uses relaxations to prune subproblems that cannot improve the incumbent solution (Land & Doig, 1960; Wolsey, 1998). Modern MILP solvers implement a richer branch-and-cut framework, where presolving, cutting planes, primal heuristics, conflict analysis, and tree-search rules interact throughout the solution process (Achterberg, 2009; Gamrath et al., 2020).

Among these components, branching and node selection are especially central because they determine how the BnB tree is generated and explored. Classical branching rules include strong branching, pseudocost branching, and reliability branching, while node-selection rules include depth-first, best-bound, best-estimate, and hybrid search strategies (Achterberg et al., 2005; Linderoth & Savelsbergh, 1999). Prior computational studies show that these rules can have a large impact on both tree size and solution time. Our work follows this solver-centered perspective, but treats branching and node selection as coupled decisions over a shared evolving tree rather than as independent heuristic modules.

A.3 Learning Branching and Node Selection Policies

Learning-based MILP research has mainly focused on replacing or approximating individual BnB decisions, reflecting a broader trend of integrating machine learning with combinatorial optimization solvers (Lodi & Zarpellon, 2017; Bengio et al., 2021). For branching, early approaches learn variable scores or rankings from strong-branching demonstrations using hand-crafted features (Khalil et al., 2016; Alvarez et al., 2017). Later work represents the LP relaxation as a variable–constraint bipartite graph and uses graph neural networks to imitate strong branching (Gasse et al., 2019). Several studies further incorporate search-tree state or trajectory information into branching, for example by parameterizing the BnB tree state (Zarpellon et al., 2021), exploiting parent–child dependencies in branching labels (Gupta et al., 2022), or learning branching through reinforcement learning and retrospective trajectories (Etheve et al., 2020; Parsonson et al., 2023). More recently, T-BranT integrates branching history through a tree-aware transformer (Lin et al., 2022), while Mamba-Branching models sequences of branching states and decisions (Zhang et al., 2026).

For node selection, He et al. (2014) apply imitation learning to learn adaptive BnB search orders, while Labassi et al. (2022) use a siamese graph neural network to compare open nodes. TRGNN represents the evolving BnB tree as a tripartite graph for node selection

(Zhang et al., 2025). These methods provide strong evidence that BnB decisions can be learned from data. However, they typically optimize either branching or node selection in isolation. Seq-BnB is complementary to these works: it uses a bipartite graph encoder for local LP structure, but conditions both branching and node-selection heads on the same global sequence representation.

A.4 Learning Other Solver Components and Global Search Context

Beyond branching and node selection, recent work has applied learning to other solver components. Neural diving and related primal-learning approaches predict high-quality assignments or partial assignments to accelerate feasible-solution discovery (Nair et al., 2020; Sonnerat et al., 2021). Learning-based heuristic scheduling aims to decide when and how to call primal heuristics inside BnB (Chmiela et al., 2021). Cut selection has also attracted increasing attention, with imitation learning, reinforcement learning, and survey work showing that the choice of cuts can be treated as a learnable solver decision (Paulus et al., 2022; Turner et al., 2023; Deza & Khalil, 2023).

Recent studies have further moved beyond isolated solver decisions by exploring shared representations across tasks or coupled policies within BnB. Cai et al. (Cai et al., 2025) proposed a multi-task representation learning framework that learns shared MILP embeddings across backdoor identification, predict-and-search, and solver configuration, while using task-specific output heads for different downstream decisions. From a broader generalization perspective, Li et al. (Li et al., 2025) investigated a foundation-model paradigm for MILP, training a single model on a large and diverse collection of generated MILP classes for integrality-gap prediction, learning to branch, and language–MILP alignment. More closely related to the coupling of BnB decisions, Du et al. (Du et al., 2025) developed a unified learning-based framework that jointly integrates node selection and variable selection, highlighting the interdependence between these two decisions during the search process.

These works suggest that a general learning interface for MILP solving should not be restricted to a single solver component. Nevertheless, existing multi-task and foundation-model approaches primarily focus on representations shared across tasks or problem classes; history-aware branching and tree-aware node-selection methods address one decision type at a time, while prior joint BnB policies do not study a shared chronological sequence of both node-selection and branching events. In contrast, Seq-BnB represents an evolving BnB run as a typed sequence of node and action tokens and combines this search-history representation with the current MILP bipartite graph. This design enables branching and node selection to condition on a shared global search context, rather than relying only on local solver states or static instance-level embeddings. Although the present work focuses on branching and node selection, the same sequence-based interface could be extended to incorporate additional solver actions, such as cut selection, heuristic scheduling, or other learned control decisions.

B Token Features and Online Information

Table 4 summarizes when each feature group becomes available and how it is used.

C Expert Oracle and Training Details

Oracle labels for node selection. For each training instance, we first solve the MILP to optimality offline and fix one reference optimal solution $x^\star$. The reference solution is used only for producing training labels and is never provided to Seq-BnB at test time.

Each BnB node u defines a local domain through the branching decisions on the path from the root to u. Let $L_j(u)$ and $U_j(u)$ denote the lower and upper bounds imposed on integer variable x_j at node u, including both the original bounds and all branching-bound changes along the path. We say that node u is compatible with the reference solution $x^\star$ if

$$L_j(u) \le x_j^\star \le U_j(u), \qquad \forall j \in \mathcal{I}.$$

Table 4: Information availability in the online construction of Seq-BnB.

Token or Quantity	Feature Group	Availability	Use in Scoring
Open-node token	Stored node bound $\ell(u)$, estimate $c(u)$, depth $d(u)$, creation/visit order $s(u)$, and currently available global statistics	Available before node selection from the solver's open-node data structure; no LP relaxation is solved solely for scoring open nodes	Scores node selection through Eq. equation 13
Processed-node update	Visited, fathomed, and incumbent-update indicators, together with realized node status	Available only after the corresponding node has been processed or fathomed, or after an incumbent update has occurred	Past context only
Node-selection action token	Selected-node statistics $d(n_t)$, $\ell(n_t)$, $c(n_t)$, and the local summary $\mathbf{s}^{\text{node}}(n_t)$ after the selected node is processed	Created after the node-selection decision; $\mathbf{s}^{\text{node}}(n_t)$ is not available before selecting n_t	Not used to choose n_t; used only for branching at the processed node and future decisions
Branching candidates	Local variable embeddings $\mathbf{h}_j^{\text{var}}(n_t)$ for fractional candidate variables	Available after solving the LP relaxation at the selected node n_t	Scores branching through Eqs. equation 11 and equation 12
Branching action token	Selected-variable embedding $\mathbf{h}_{j_t}^{\text{var}}(n_t)$ and current search statistics	Created after the branching variable has been selected	Future context only

Equivalently, $x^\star$ satisfies all branching decisions that define node u. At node-selection state t, define the compatible open-node set

$$\mathcal{C}_t(x^\star) = \{u \in \mathcal{O}_t : L_j(u) \le x_j^\star \le U_j(u),\ \forall j \in \mathcal{I}\}.$$

If $\mathcal{C}_t(x^\star)$ is nonempty, the oracle selects a compatible node according to a deterministic tie-breaking rule:

$$u_t^\star = \text{lexmin}_{u \in \mathcal{C}_t(x^\star)}\big(-d(u),\ \ell(u),\ c(u),\ s(u)\big).$$

Thus, among all open nodes that can still contain the reference optimum, the oracle first prefers the deepest node, then the smaller LP lower bound, then the smaller estimate value, and finally the earlier creation order. This rule follows a path toward the fixed optimal solution while keeping the label deterministic when several compatible nodes exist.

If $\mathcal{C}_t(x^\star)$ is empty, which can occur after the reference-optimal path has already been processed or when a learner-induced state no longer contains a compatible open node, the oracle falls back to a deterministic proof-oriented rule:

$$u_t^\star = \text{lexmin}_{u \in \mathcal{O}_t}\big(\ell(u),\ c(u),\ -d(u),\ s(u)\big).$$

That is, it selects the open node with the smallest lower bound, breaking ties by estimate value, depth, and creation order. The compatibility test only uses the branching bounds that define each open node and the fixed reference solution $x^\star$; it does not require future node outcomes or LP information from unprocessed open nodes. Therefore, the oracle label is well-defined not only on expert-generated trees, but also on learner-induced trees encountered during DAgger.

C.1 Candidate-Set Losses

For a branching sample i, let $\mathcal{V}_i$ be the candidate set and let a_i^* be the expert variable. The branching loss is

$$\mathcal{L}_{\text{branch}} = -\frac{1}{|\mathcal{B}|}\sum_{i \in \mathcal{B}} \log \frac{\exp(\hat{y}_{a_i^*}^{\text{branch}})}{\sum_{j \in \mathcal{V}_i} \exp(\hat{y}_j^{\text{branch}})}. \tag{17}$$

For a node-selection sample i, let $\mathcal{O}_i$ be the open-node set and $u_i^\star$ be the oracle node defined by the compatibility rule in Section 4.1. The node-selection loss is

$$\mathcal{L}_{\text{ns}} = -\frac{1}{|\mathcal{I}_{\text{ns}}|}\sum_{i \in \mathcal{I}_{\text{ns}}} \log \frac{\exp(\hat{y}_{u_i^*}^{\text{ns}})}{\sum_{u \in \mathcal{O}_i} \exp(\hat{y}_u^{\text{ns}})}. \tag{18}$$

This loss is aligned with inference, where the solver selects the maximum-scoring candidate from the relevant candidate set.

Table 5: Problem instance statistics for test and transfer sets.

Statistic	Set Covering		Comb. Auction		Facility Location		Max Ind. Set	
	Test	Transfer	Test	Transfer	Test	Transfer	Test	Transfer
Avg. binary variables	1000.00	1000.00	500.00	1000.00	100.00	200.00	500.00	1000.00
Avg. integer variables	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Avg. continuous variables	0.00	0.00	0.00	0.00	10000.00	40000.00	0.00	0.00
Avg. constraints	500.00	1000.00	191.90	386.10	10201.00	40401.00	2088.60	4219.30
Instances	100	100	100	100	100	100	100	100

All values are averages over the corresponding set.

Table 6: Results on standard test instances.

Method	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
SB+Oracle	9.01	20.89	–	59.34	183.28	–	82.99	131.58	–	101.37	30.44	–
PSB+Estimate	3.73	37.25	8	10.98	886.54	9	41.33	226.81	1	9.88	101.93	6
RankNet+PSB	3.87	37.49	12	16.52	1247.04	1	28.70	256.24	28	11.42	86.91	4
L2C+PSB	3.60	38.86	12	29.75	1806.76	0	30.80	265.38	21	7.78	156.95	15
L2B+Estimate	2.99	32.06	24	10.13	292.23	49	28.60	180.19	30	6.45	38.10	27
Retro+Estimate	6.36	136.16	0	18.44	1124.61	0	35.36	230.98	0	12.27	144.39	0
Seq-BnB	2.81	28.62	44	10.61	252.34	41	30.22	164.16	20	6.35	36.54	48

Time is in seconds. Time and node counts are shifted geometric means. Bold values mark the best nonexpert method.

D Instance Statistics

Table 5 lists the instance sizes and sample counts for the standard and larger transfer sets used in Section 5.2.

E Complete Main Results

Tables 6 and 7 provide the full standard-test and transfer results underlying Section 5.2, including the win counts omitted from the compact main-text tables. A win records the shortest solving time on an instance among nonexpert methods; SB+Oracle is an expert reference.

F Detailed Ablations

Tables 8, 9, and 10 give the full results for the sequence-context, joint-learning, and DAgger ablations summarized in Section 5.3. In the w/o Seq variant, the branching head uses only the local variable embedding $\mathbf{h}_j^{\text{var}}(n_t)$. Figure 2 shows branching imitation accuracy with and without sequence context.

G Hyperparameter Sensitivity

G.1 Sequence Length and Loss Coupling

We further examine two hyperparameters that are directly tied to the proposed formulation: the maximum sequence window length $L_{\max}$ and the joint-loss weight λ. Unlike generic training hyperparameters, these two quantities have clear algorithmic meanings. The window length $L_{\max}$ controls how much recent BnB history is retained by the global sequence encoder, while λ controls the relative strength of node-selection supervision in the shared representation.

Table 11 reports the effect of varying $L_{\max}$. A larger window provides the self-attention encoder with more search-history information, but also increases memory usage and attention cost. The results show a consistent improvement as $L_{\max}$ increases from 100 to 500. Across all four benchmark families, $L_{\max} = 500$ gives the lowest shifted geometric mean in both solving time and processed nodes, and it wins the largest number of instances among the three tested values. This supports the importance of retaining sufficient global search

Table 7: Results on transfer instances.

Method	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
SB+Oracle	121.58	234.79	–	3363.43	2074.32	–	2304.62	207.62	–	2795.54	183.65	–
PSB+Estimate	25.66	667.39	43	2174.09	21105.59	0	748.19	1390.19	14	1217.84	13517.28	1
RankNet+PSB	40.38	802.18	1	1228.58	37844.01	1	2113.60	4398.48	0	644.66	11997.45	4
L2C+PSB	45.72	696.74	2	1526.97	66897.30	0	1407.08	3611.90	0	601.47	16559.32	5
L2B+Estimate	28.97	462.74	31	618.79	11771.63	72	747.34	1475.62	16	459.58	6085.88	17
Retro+Estimate	514.17	7766.10	0	3591.17	27588.27	0	1162.38	1740.44	1	2171.13	18567.67	0
Seq-BnB	37.33	402.63	23	790.99	5134.41	27	621.68	1249.23	69	412.42	1179.97	73

Time is in seconds. Time and node counts are shifted geometric means. Bold values mark the best nonexpert method.

Table 8: Effect of sequence context on standard test instances.

Method	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
Seq-BnB (w/o Seq)	3.51	31.94	18	11.83	281.63	21	41.56	238.42	15	7.48	42.73	7
Seq-BnB	2.81	28.62	82	10.61	252.34	79	30.22	164.16	85	6.35	36.54	93

Wins are counted between the two compared methods.

context: Seq-BnB benefits not only from the currently open nodes, but also from recent tree-growth and decision-history information.

Table 12 reports the effect of the joint-loss weight λ. When λ is too small, the shared encoder receives weak supervision from the node-selection task; when λ is too large, the node-selection loss can dominate the branching objective and bias the shared representation. The results exhibit this tradeoff clearly. Intermediate values perform better than the two extremes, and the default value $\lambda = 0.1$ achieves the best time and node count on all four benchmark families. These results indicate that joint learning is most effective when branching and node selection are balanced, rather than when one task dominates the shared encoder.

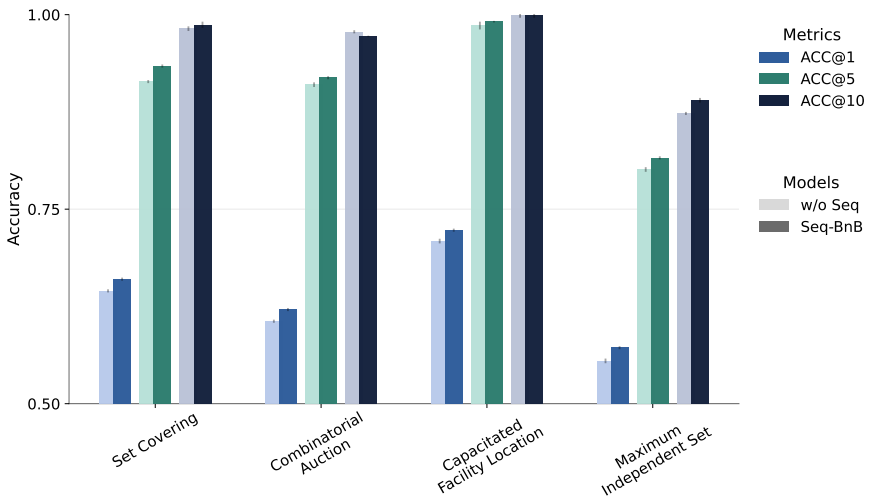


Figure 2: Branching imitation accuracy with and without sequence context. ACC@k measures whether the expert variable is among the top k predicted candidates. Sequence context improves top-k accuracy on most benchmarks; together with Table 8, this suggests that modest gains in single-step imitation can translate into larger improvements over the subsequent BnB search.

Table 9: Joint learning ablation

Method	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
Seq-BnB (BR-only)	3.04	34.24	12	12.18	306.54	11	39.56	225.63	3	6.92	39.84	15
Seq-BnB (NS-only)	3.80	32.25	6	27.47	684.73	3	40.88	236.94	5	7.89	68.74	10
Seq-BnB	2.81	28.62	82	10.61	252.34	86	30.22	164.16	92	6.35	36.54	75

Table 10: Effect of DAgger data aggregation.

Method	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
Seq-BnB (w/o DAgger)	5.36	43.21	5	18.21	397.63	0	49.37	374.35	2	8.64	49.51	4
Seq-BnB	2.81	28.62	95	10.61	252.34	100	30.22	164.16	98	6.35	36.54	96

Wins are counted between the two compared methods.

Table 11: Effect of maximum sequence-window length $L_{\max}$ on standard test instances.

$L_{\max}$	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
100	3.46	34.87	5	13.28	334.52	1	39.84	232.65	0	7.58	45.37	0
300	3.07	31.46	8	11.76	287.91	4	34.92	193.74	6	6.91	39.82	2
500	2.81	28.62	87	10.61	252.34	95	30.22	164.16	94	6.35	36.54	98

Wins are counted among the three window sizes.

Table 12: Effect of joint-loss weight λ on standard test instances.

λ	Set Covering			Comb. Auction			Facility Location			Max Ind. Set		
	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins	Time	Nodes	Wins
0.01	3.23	32.84	17	11.42	294.18	22	35.76	206.43	18	6.84	40.43	22
0.05	2.96	30.15	24	10.97	265.83	27	31.48	178.92	27	6.57	37.88	28
0.1	2.81	28.62	35	10.61	252.34	33	30.22	164.16	34	6.35	36.54	34
0.5	3.18	31.76	16	12.64	321.57	12	33.95	191.35	14	7.12	43.16	11
1.0	3.59	35.02	8	14.28	402.91	6	37.64	224.78	7	7.64	51.28	5

Wins are counted among the five values of λ.